SSCM Exercise 9

Nikolaus Czernin - 11721138

```
library("tidyverse")
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library("knitr")
library("mcmcplots")
```

```
## Loading required package: coda
```

```
library("gridExtra")
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

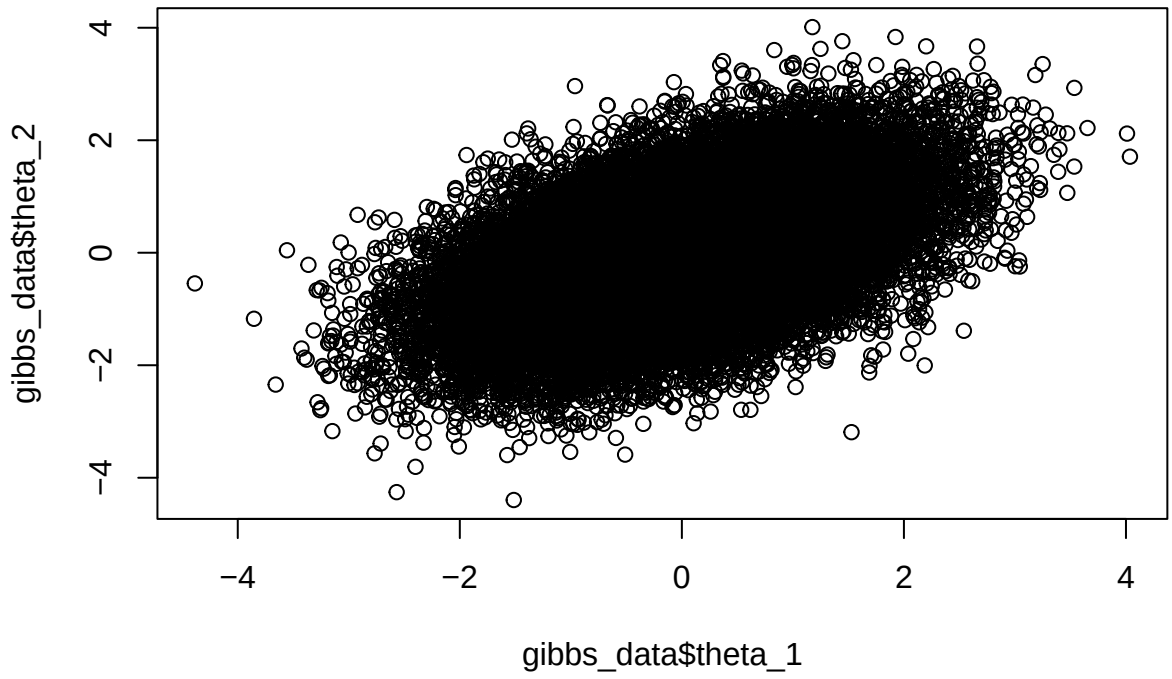
```
# Custom print function
print_ <- function(...) print(paste(...))

set.seed(11721138)
```

```
# parameters of the bivariate distribution
means <- c(0, 0)
p <- 0.5 # this is actually "rho", but I use p because its easier to spot
covariances <- matrix(c(1, p, p, 1), nrow = 2)
# Markov chain size
M <- 30000
```

Gibbs Sampling

We iteratively sample θ_1 given θ_2 (and vice versa)

```
gibbs_sampler <- function(M, rho){
  # create empty number variables for the parameters, of size M
  theta_1 <- numeric(M)
  theta_2 <- numeric(M)
  # initialize random starting values
  theta_1 <- rnorm(1)
  theta_2 <- rnorm(1)
  # make the following M-1 iterations
  # use t as iterating variable, like in the slides
  for (t in 2:M){
    # sample using the conditional distributions
    # sample theta_1 given theta_2, save the result as the ith value
    theta_1[t] <- rnorm(1, mean = p * theta_2[t-1], sd = sqrt(1 - p^2))
    # sample theta-2 given theta_1, ...
    theta_2[t] <- rnorm(1, mean = p * theta_1[t], sd = sqrt(1 - p^2))
  }
  data.frame(theta_1 = theta_1, theta_2 = theta_2)
}
gibbs_data <- gibbs_sampler(M, p)
plot(gibbs_data$theta_1, gibbs_data$theta_2)
```



Here I create a Gibbs sampling algorithm. It creates a sample of size M for all wanted parameters (in our

case 2). It initializes the sample witha random value of standard normal distribution. Then it iteratively updates both parameters, using the existing sample of the other respective parameter as a conditional.

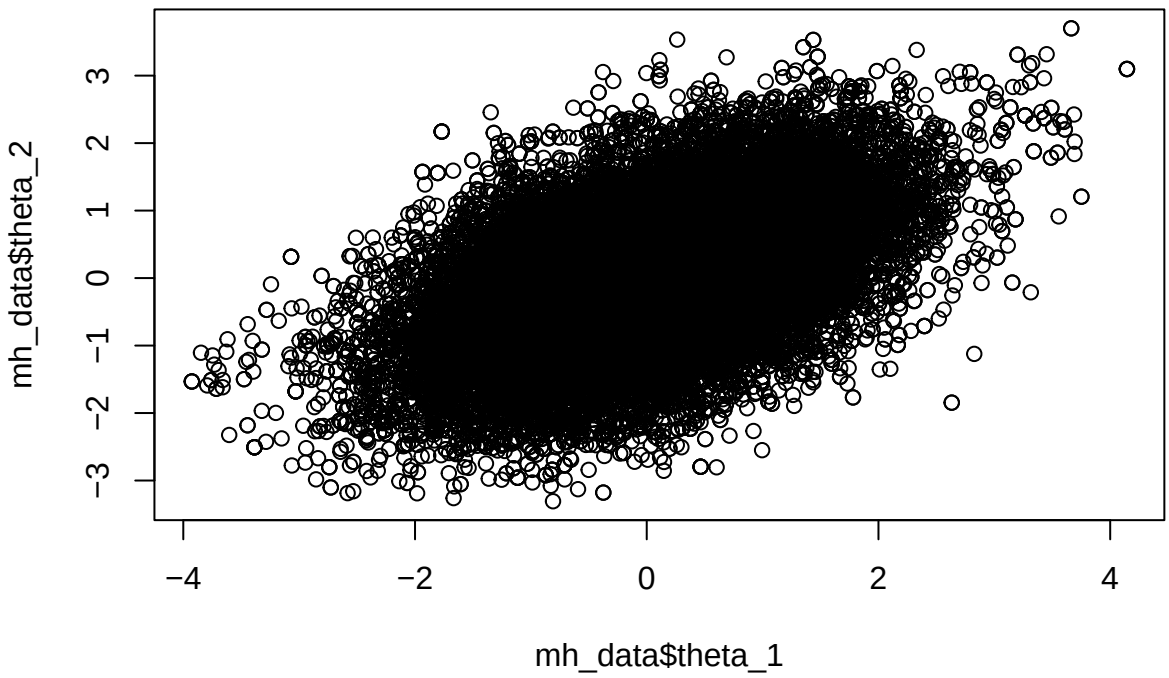
```
target_pdf <- function(theta_1, theta_2, rho){
  # TODO: I think the first fraction can be omitted because it is contant
  # for all samples, but Im not sure so just leave it
  1 / (2*pi*sqrt(1-rho^2)) * exp(-1 / (2*(1-rho^2)) * (theta_1^2 - 2*rho*theta_1 * theta_2 + theta_2^2))
}
```

This function is the probability density function of th etask.

```
metropolis_hastings <- function(M, rho){
  # create empty theta 1 and theta 2
  theta_1 <- numeric(M)
  theta_2 <- numeric(M)
  # their first values are random
  theta_1[1] <- rnorm(1)
  theta_2[1] <- rnorm(1)
  print(theta_1[1] == theta_2[1])
  # iterate over the remaining M-1 sampels
  for (t in 2:M){
    # generate candidate thetas
    candidate_1 <- rnorm(1, mean = theta_1[t-1], sd = 0.5)
    candidate_2 <- rnorm(1, mean = theta_2[t-1], sd = 0.5)
    # compuite the probability of accepting the candidate
    alpha <- target_pdf(candidate_1, candidate_2, rho) /
      target_pdf(theta_1[t-1], theta_2[t-1], rho)
    # generate u from uniform distribution
    u <- runif(1, 0, 1)
    if (u < alpha){
      # accept the candidates as new parameters
      theta_1[t] <- candidate_1
      theta_2[t] <- candidate_2
    } else {
      # keep the previous parameters
      theta_1[t] <- theta_1[t-1]
      theta_2[t] <- theta_2[t-1]
    }
  }
  # return all parameters
  data.frame(theta_1 = theta_1, theta_2 = theta_2)
}


mh_data <- metropolis_hastings(M, p)
```

```
## [1] FALSE
```

```
# Plot samples
plot(mh_data$theta_1, mh_data$theta_2)
```



Here I implement the Metropolis-Hastings algorithm to simulate the distribution from the task. I, again, start with 2 empty vectors of size M and randomly initialize the first parameters from the standard normal distribution. Then I iteratively generate new candidates by randomly deviating from the previous iteration's parameters. Then i compute an acceptance ratio from the probability density function defined before. Like in the acceptance-rejection method, I generate a random uniform variable and use it to determine, whether the new parameter candidate ratio makes them eligible and based on this decision I update this iteration's parameters.

Diagnostics

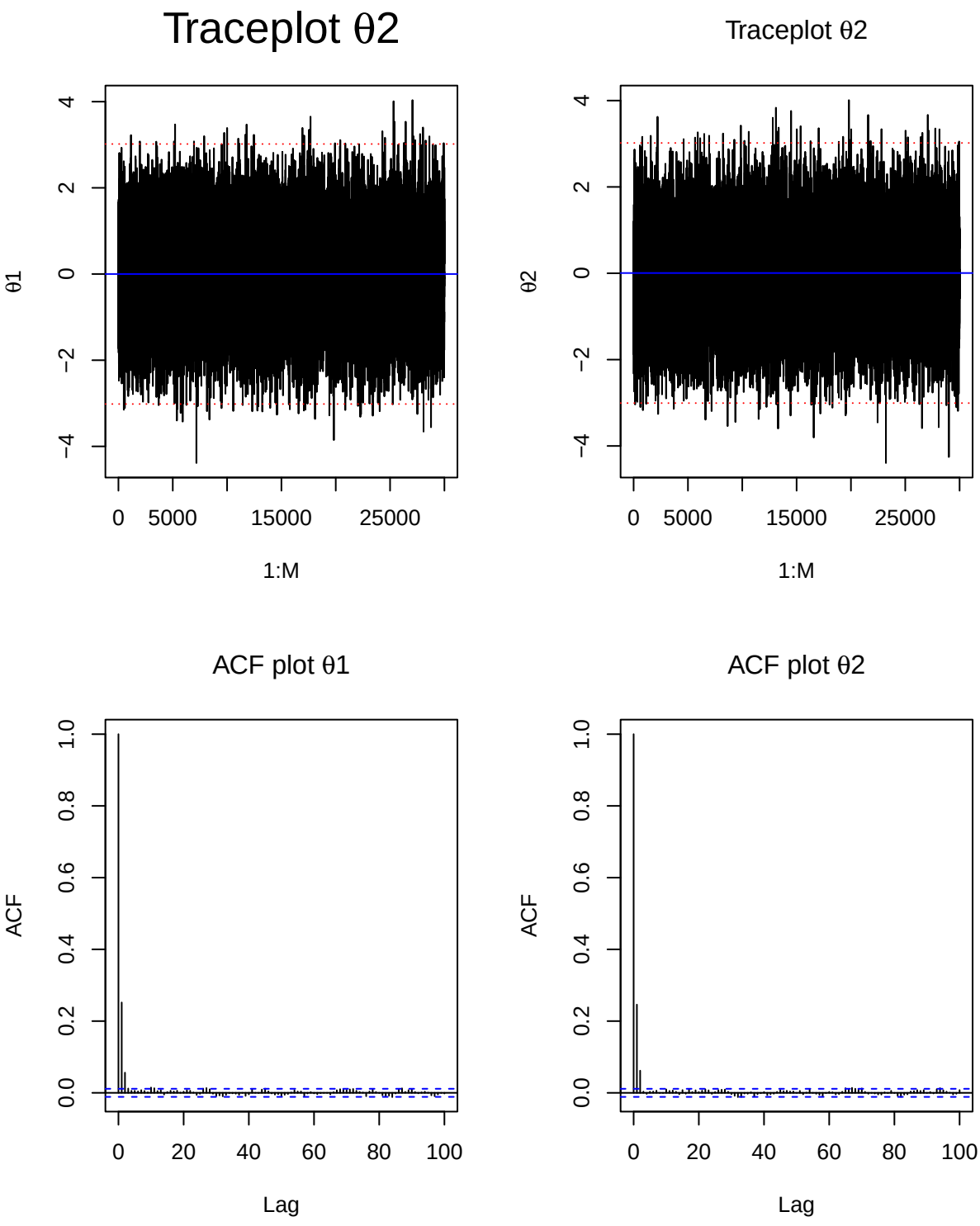
```
diagnostics <- function(t1, t2){
  par(mfrow=c(2, 2))
  plot(1:M, t1, type="l",
       ylim=c(min(t1,mean(t1)-3*sd(t1)),
              max(t1,mean(t1)+3*sd(t1))),
       ylab=expression(paste(theta,"1")),
       cex.main=2,
       main=expression(paste("Traceplot ",theta,"2")))
  abline(h=mean(t1), col="blue");
  abline(h=mean(t1)+3*sd(t1), lty=3, col="red")
  abline(h=mean(t1)-3*sd(t1), lty=3, col="red")

  plot(1:M, t2, type="l",
       ylim=c(min(t2,mean(t2[-(1:200)]) - 3*sd(t2[-(1:200)])),
```

```
            max(t2, mean(t2[-(1:200)]) + 3*sd(t2[-(1:200)]))),
      ylab=expression(paste(theta,"2")),
      main=expression(paste("Traceplot ",theta,"2")))
  abline(h=mean(t2[-(1:200)]), col="blue")
  abline(h=mean(t2[-(1:200)]) + 3*sd(t2[-(1:200)]), lty=3, col="red")
  abline(h=mean(t2[-(1:200)]) - 3*sd(t2[-(1:200)]), lty=3, col="red")

  # ACF plots with run mean
  acf(t1, lag.max=100, main=expression(paste("ACF plot ",theta,"1")))
  acf(t2, lag.max=100, main=expression(paste("ACF plot ",theta,"2")))


  # histrograms of marginal posteriors
  m1 <- rmeanplot(t1, main=expression(paste("Run mean for ", theta, "1")))
  m2 <- rmeanplot(t2, main=expression(paste("Run mean for ", theta, "2")))


  par(mfrow=c(1,2),mar=c(5,5,4,2))
  hist(t1, freq=F, col="grey", main="Histogram", xlab=expression(paste(theta,"1")))
  curve(dnorm(x,0,1), from=-4, 4, add=T, col="red")
  hist(t2, freq=F, col="grey", main="Histogram", xlab=expression(paste(theta,"2")))
  curve(dnorm(x,0,1), from=-4, 4, add=T, col="red")
}
```

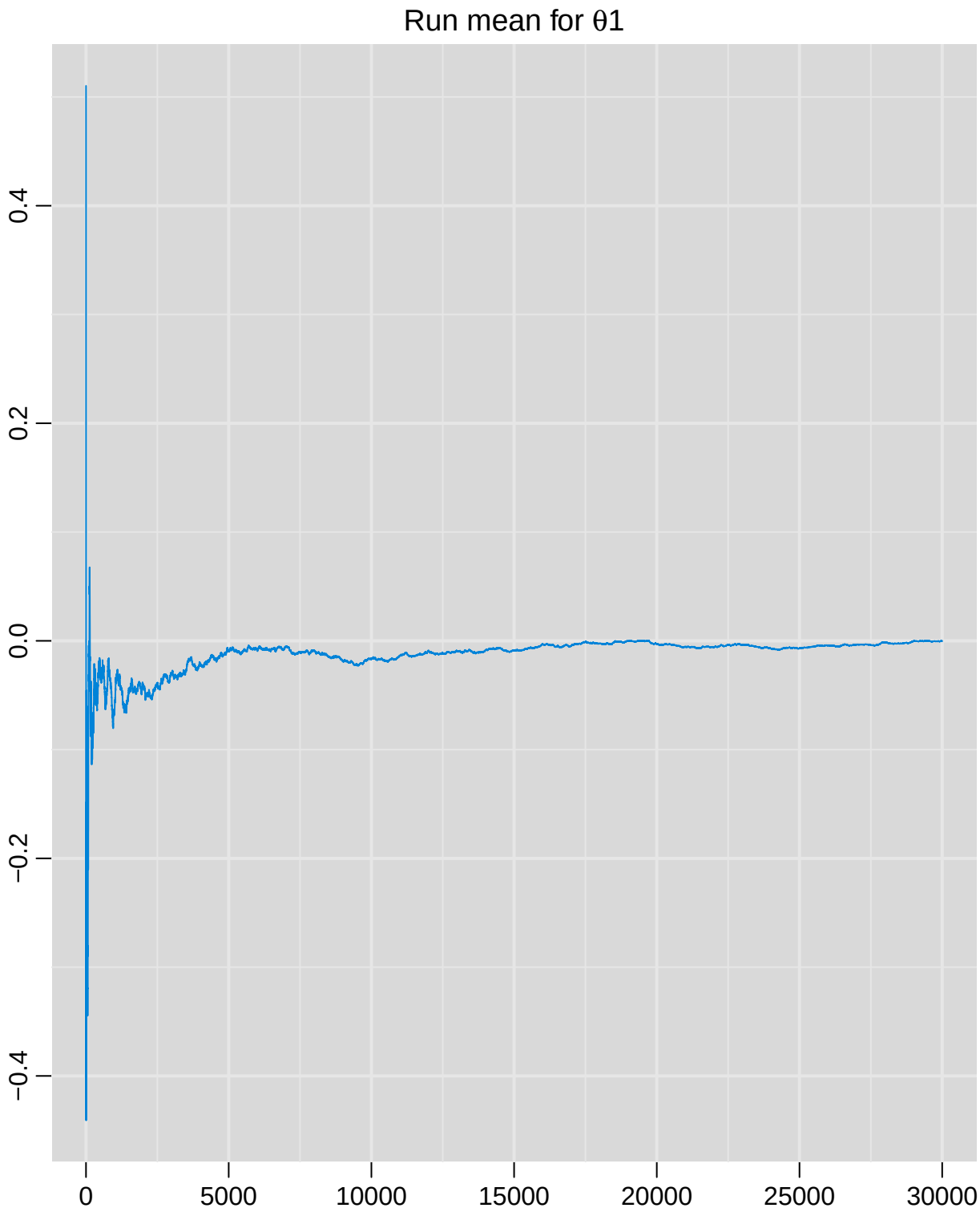
My diagnostics function creates plots that outline the behaviour of two given Markov chains. It first draws trace plots, whoch show how the parameters values evolve over the 30000 iterations. The blue horizontal lines are the mean and the red lines are the mean +/- the standard deviations * 3.
Next the functions plots the autocrrelation of the samples over different lag values. The function then plots the running means, i.e. the mean parameters for the past iterations. Finally, it plots histograms of the posterior distributions, where a standard distribution curve is overlaid to outline the central limit theorem.

Plotting the Gibbs sampler data diagnosis

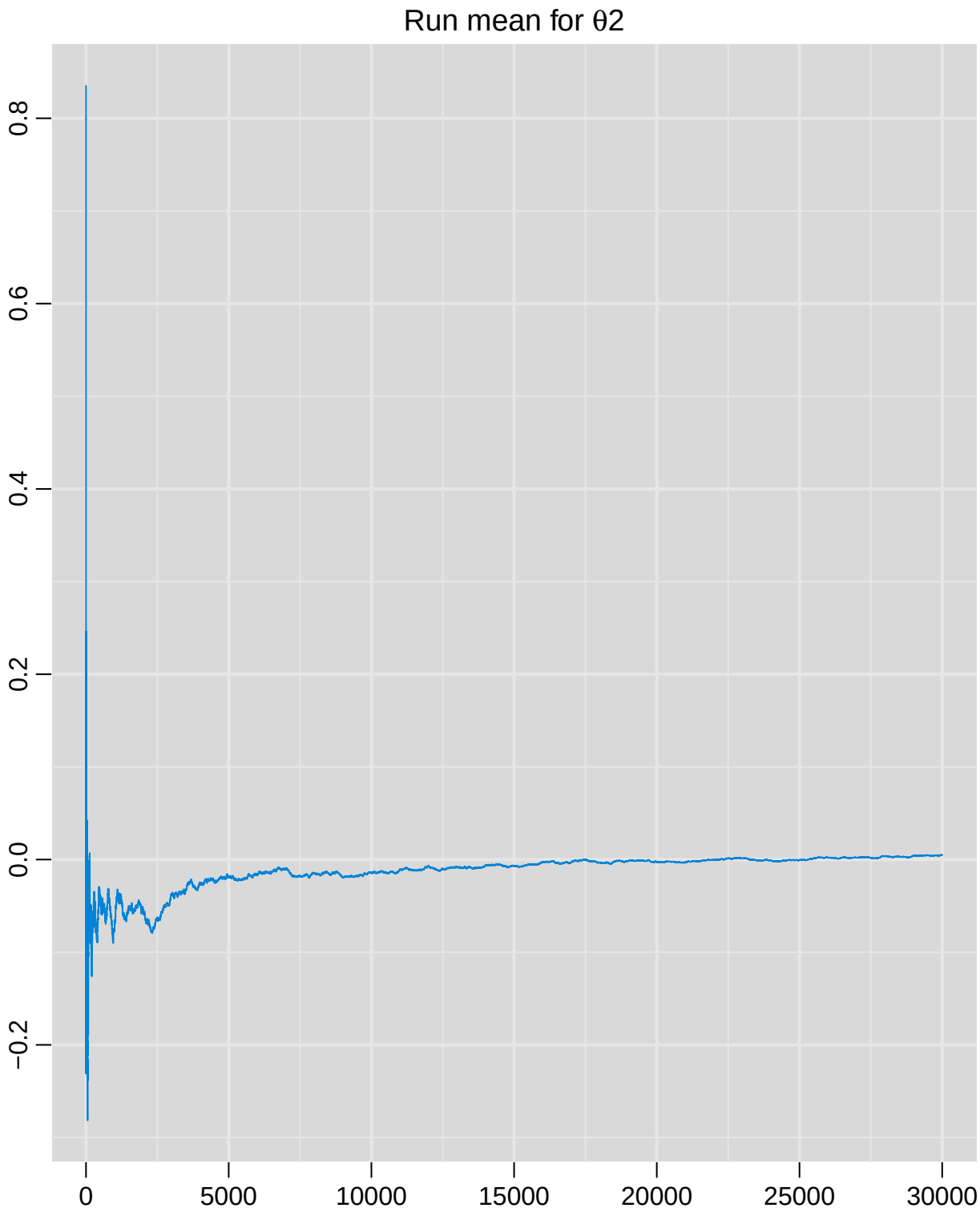
```
diagnostics(gibbs_data$theta_1, gibbs_data$theta_2)
```

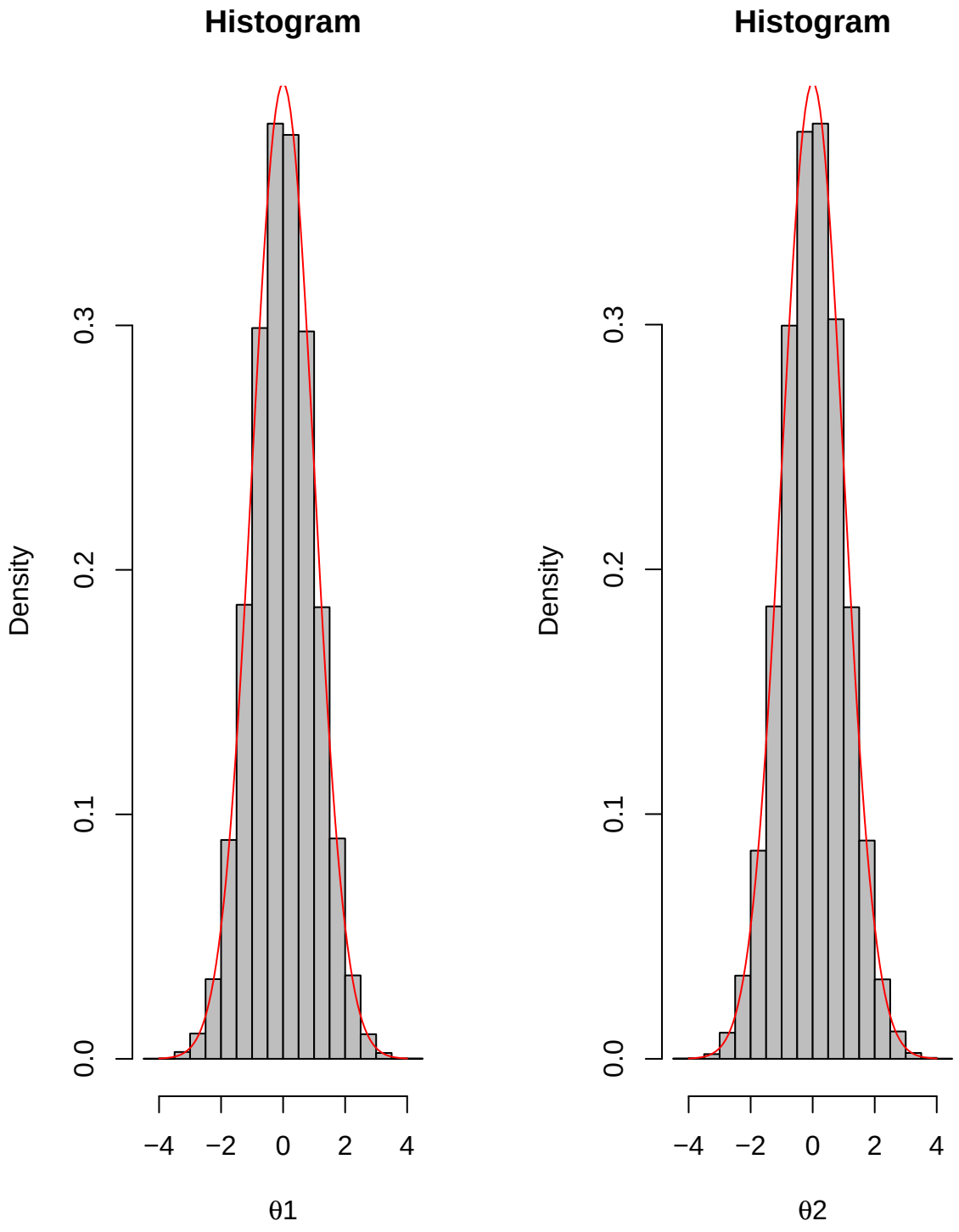



```
## Warning in rmeanplot(t1, main = expression(paste("Run mean for ", theta, :
## Argument 'mcmcout' did not have valid variable names, so names have been
## created for you.
```



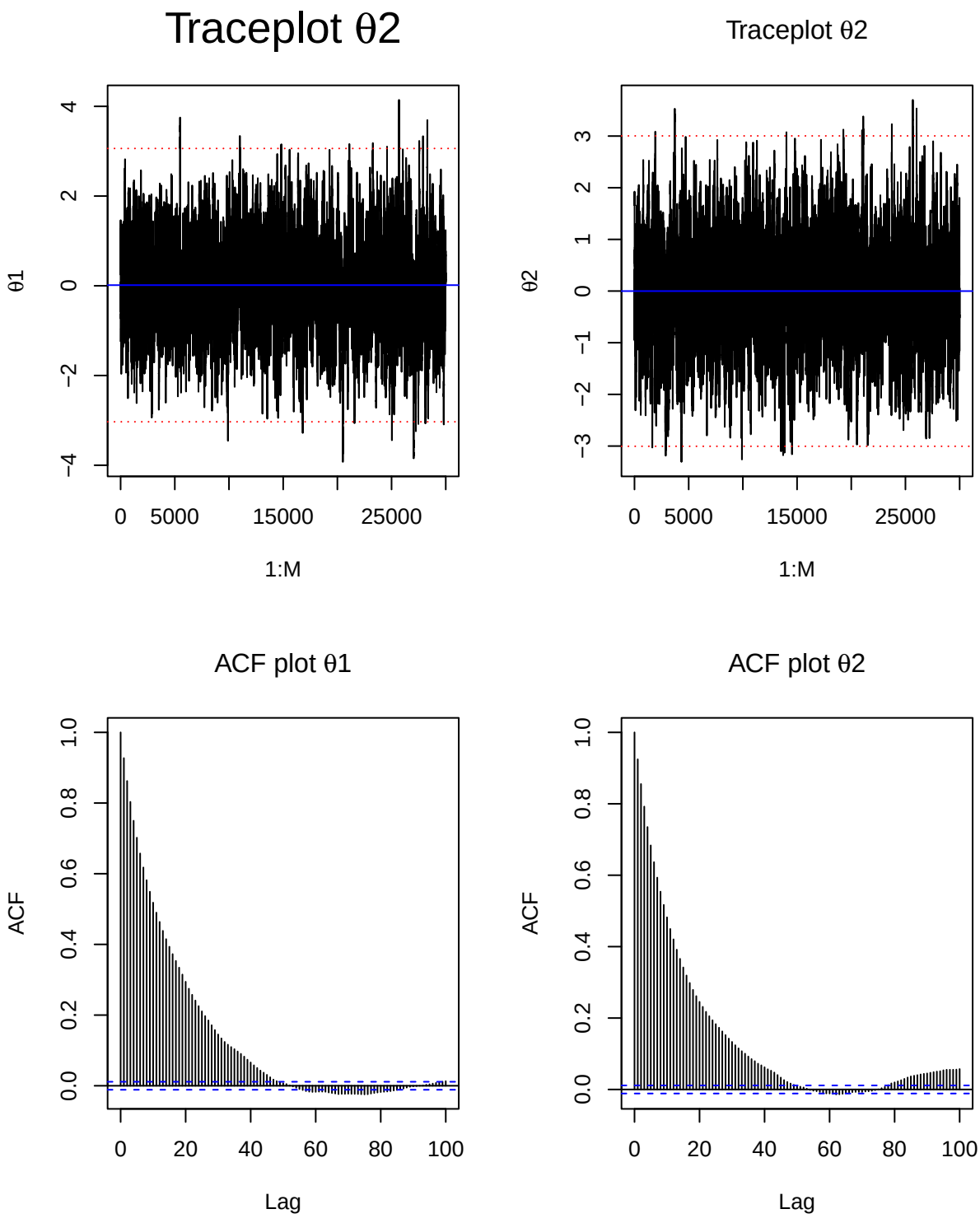

```
## Warning in rmeanplot(t2, main = expression(paste("Run mean for ", theta, :
## Argument 'mcmcout' did not have valid variable names, so names have been
## created for you.
```


Run mean for θ2
0.8
0.6
0.4
0.2
0.0
−0.2
0
5000
10000
15000
20000
25000
30000

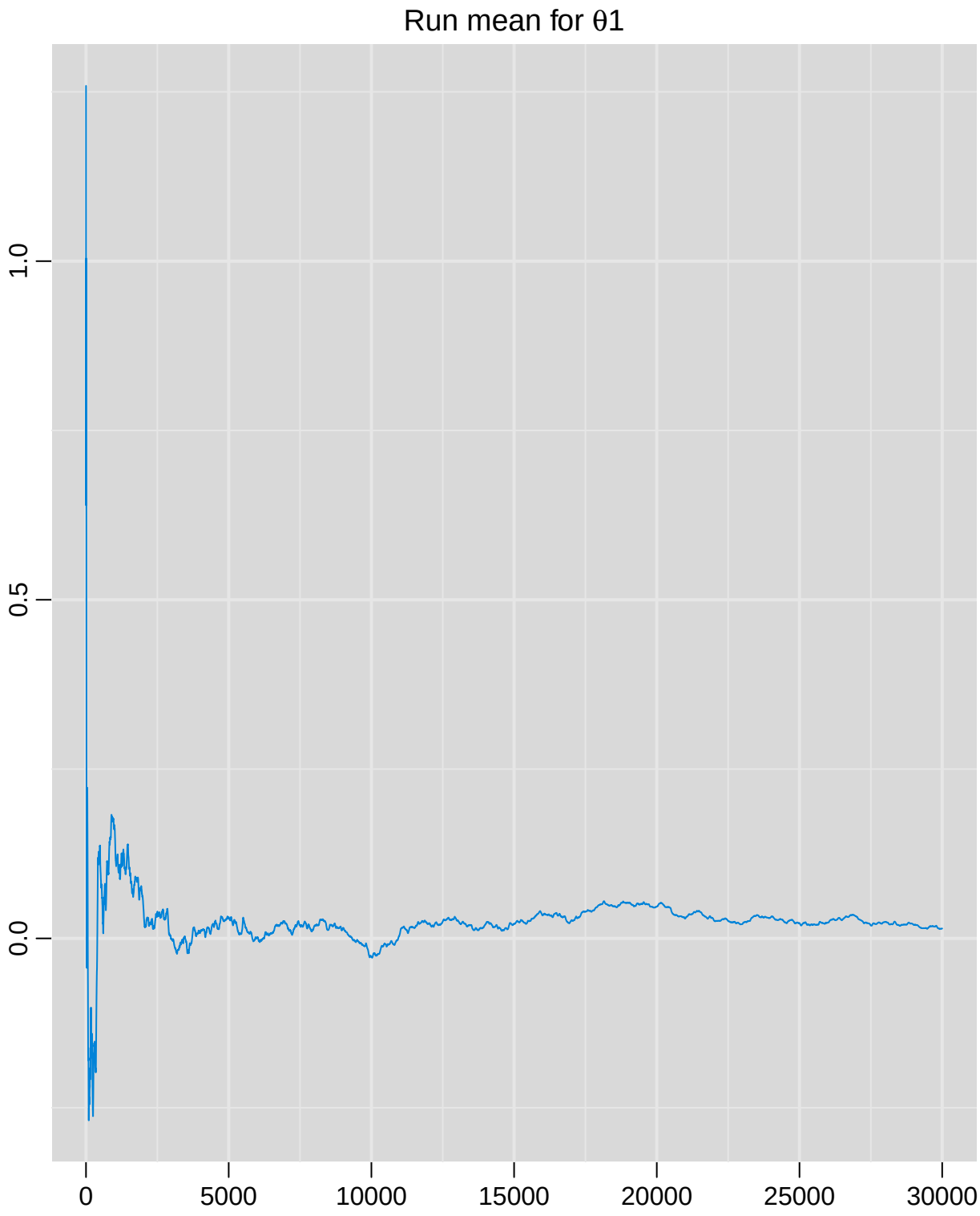


Plotting the Metropolis–Hastings data diagnosis

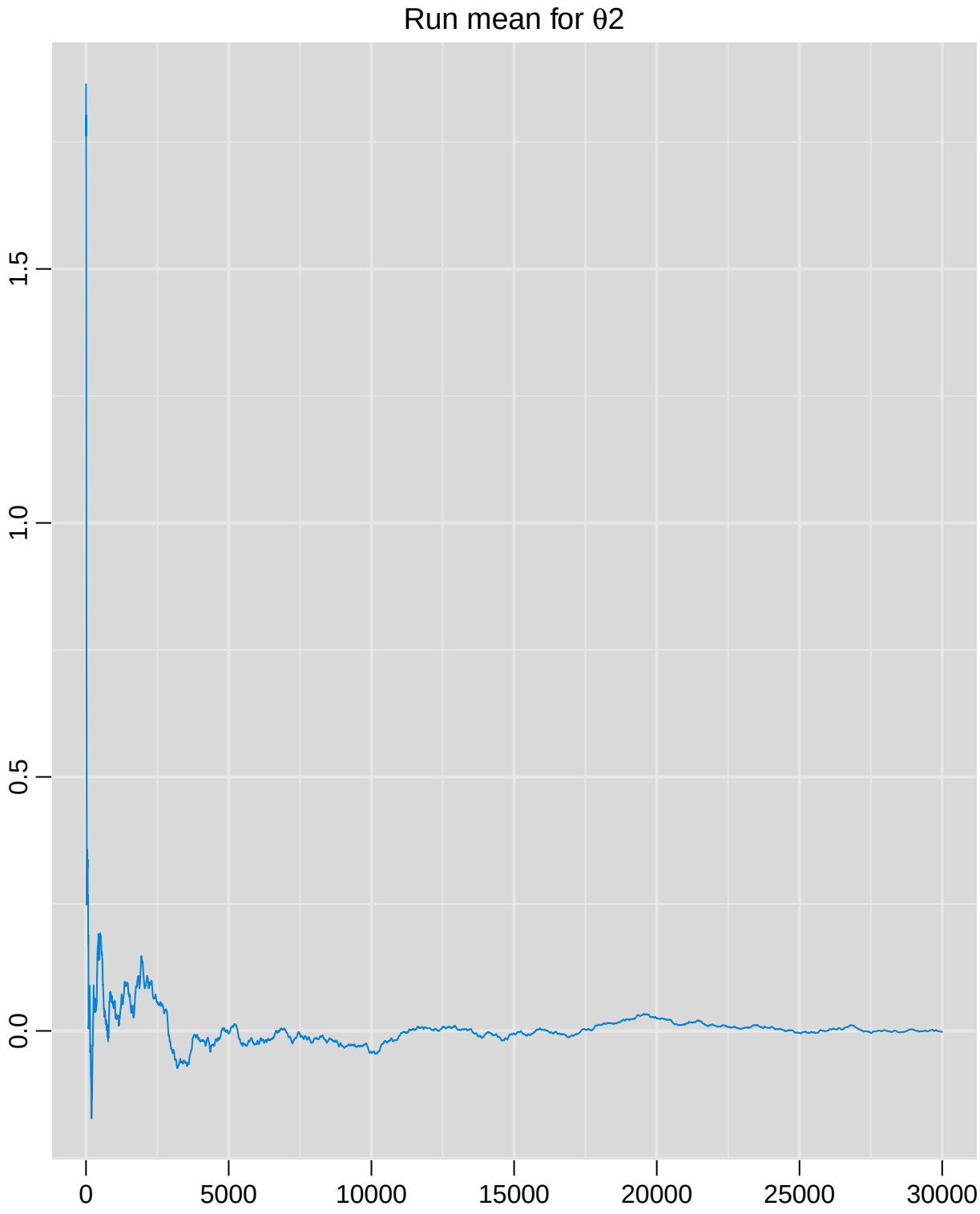
```
diagnostics(mh_data$theta_1, mh_data$theta_2)
```

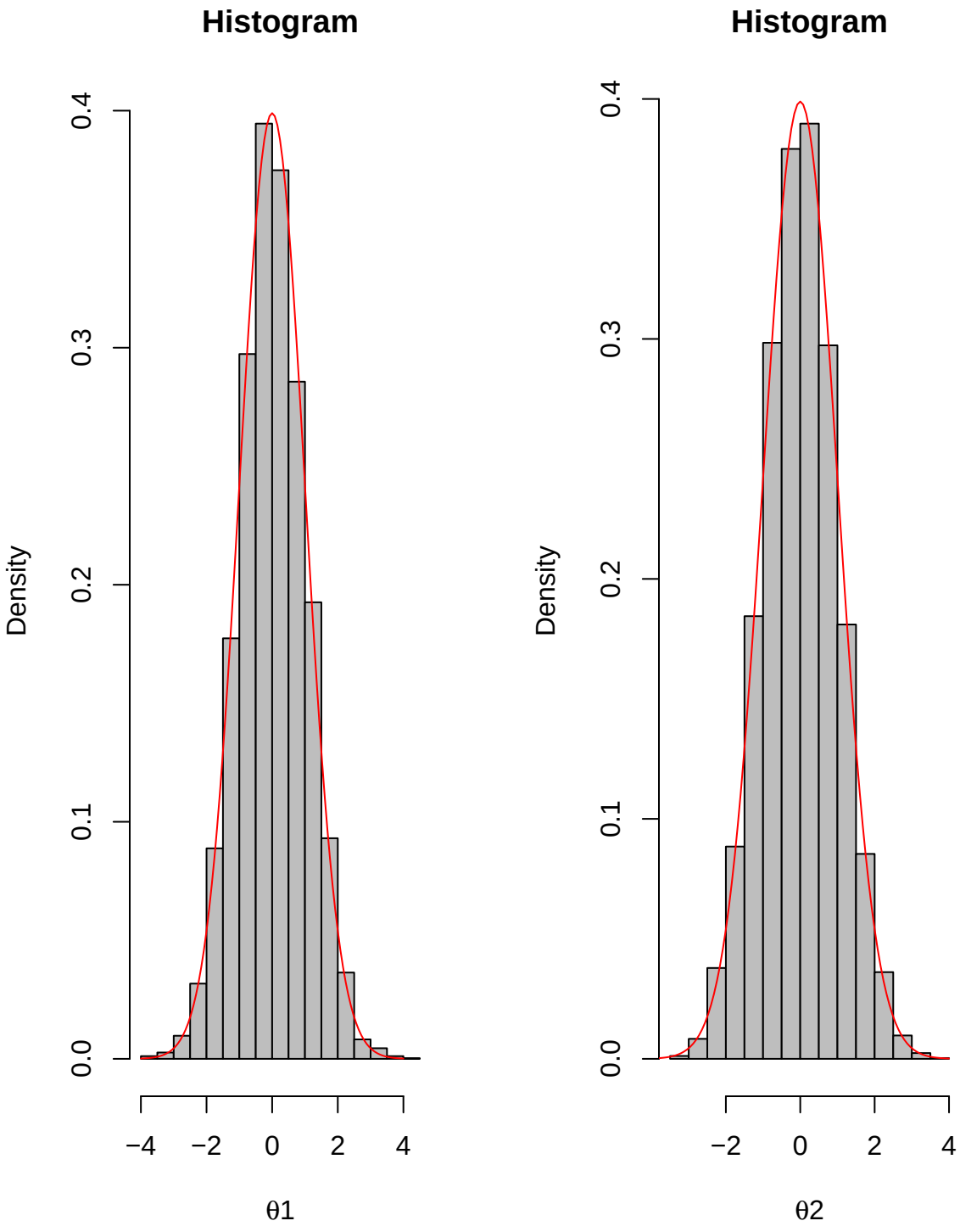



```
## Warning in rmeanplot(t1, main = expression(paste("Run mean for ", theta, :
## Argument 'mcmcout' did not have valid variable names, so names have been
## created for you.
```




```
## Warning in rmeanplot(t2, main = expression(paste("Run mean for ", theta, :
## Argument 'mcmcout' did not have valid variable names, so names have been
## created for you.
```


Run mean for θ2
1.5
1.0
0.5
0.0
0
5000
10000
15000
20000
25000
30000



In the plots of both examples, it can be observed that the parameters approach a normal distribution in the histograms. The traceplots show how stably they evolve around their means. The autocorrelation per lag value behaves differently, comparing the two sampling methods. The running mean plots show how the means become relatively stable after ~5000 iterations.